U.S. Postal Service Express Mail Label No. EE776165370US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION FOR

METHODS AND APPARATUS FOR FAULT-DETECTING AND FAULT-TOLERANT PROCESS CONTROL

Inventor:
Samuel Galpin
a citizen of the United States residing at
38 Highview Drive
Hingham, MA 02043

METHODS AND APPARATUS FOR FAULT-DETECTING AND FAULT-TOLERANT PROCESS CONTROL

Background of the Invention

The invention pertains to control systems and, more particularly, by way of non-limiting example, to fault-detecting and fault-tolerant methods and apparatus for process control.

The terms "control" and "control systems" refer to the control of the operational parameters of a device or system by monitoring one or more of its characteristics. This is used to insure that output, processing, quality and/or efficiency remain within desired parameters over the course of time.

Control is used number of fields. Process control, for example, is typically employed in the manufacturing sector for process, repetitive and discrete manufactures, though, it also has wide application in electric and other service industries. Environmental control finds application in residential, commercial, institutional and industrial settings, where temperature and other environmental factors must be properly maintained. Control is also used in articles of manufacture, from toasters to aircraft, to monitor and control device operation.

Control systems typically utilize field devices that are physically integrated into the equipment being controlled. For example, temperature sensors are usually installed directly on or within the articles, bins, or conduits that process, contain or transport the materials being measured. Control devices such as valves, relays, and the like, must also be integrated with the equipment whose operations they govern.

Predictability is among the key requirements of any control device. A fluid sensor that even occasionally produces flaky readings is unacceptable. Overengineering can insure better reliability; however, it often results in devices that are too expensive or too large for wide application.

Redundancy is a well accepted alternative to overengineering. It typically involves using two or more standard control elements in place of one. The duplicated units can be field modules, controllers or other higher-level elements in the control hierarchy.

Thus, for example, U.S. Patent 4,347,563 discloses an industrial control system in which redundant processing units serve as bus masters "of the moment," monitoring status information generated by primary processing units. If a redundant unit detects that a primary has gone faulty while executing an applications program, the redundant unit loads that program and takes over the primary's function.

U.S. Patent 5,008,805, on the other hand, discloses a real time control system in which "sender" and "listener" processors synchronously step through sequential schedules, with the sender controlling execution of events sent from a host. The listener monitors the sender and in the event of fault, assumes the role of the latter, executing commands omitted during the takeover interval.

A shortcoming of these and many other prior art redundancy schemes is their imposition of undue computational or hardware overhead. U.S. Patents 5,008,805, for example, has the disadvantage of requiring that the sender and listener operate in lock-step, necessitating common timing lines and up-front synchronization procedures.

An object of this invention is to provide improved methods and apparatus for control and more particularly, for example, for fault-tolerance and fault-detection in process control.

A related object of the invention is to provide such methods and apparatus as demand low computation and hardware overhead and thus, for example, that can be implemented on a wide range of control equipment, from field devices to plant and enterprise servers.

A further object of the invention is to provide such methods and apparatus as can be implemented in new control equipment and retrofitted into pre-existing equipment.

Summary of the Invention

The foregoing are among the objects attained by the invention which provides, in one aspect, a method of process, environmental or other control (hereinafter, "control") that includes executing a first sequence of instructions in a first process or thread (collectively, "process") and executing a second sequence of instructions in a second process. The processes are loosely coupled, that is, they do not operate in forced lock-step synchronism with one another and need not, for example, share a common clock. Though the processes can operate on the same processor, typically, they operate on different processors, e.g., within separate elements of a control system. Thus, for example, according to some aspects of the invention, the first and second processes are resident and execute on respective partners in paired field devices (e.g., sensors), block controllers, process controllers, and plant or enterprise servers.

The method further includes comparing states of the first and second processes following their respective completions of the first and second instruction sequences. The comparison can cover registers, memory, flags, interrupts, tasks, and/or events for each of the respective processes. Thus, for example, to determine whether the two processes executed their respective sequences substantially identically -- and, therefore, are ready to begin to execute additional sequences -- one aspect of the invention calls for comparing flags set during execution by the interrupt handlers of each process. Execution of further instruction sequences by either process is delayed, according to related aspects of the invention, pending a favorable comparison of the process states. To this end, one aspect of the invention calls for comparing the states multiple times, e.g., over a predefined time interval. In addition to compensating for non-significant differences in execution speed or event occurrence, employing multiple comparisons insures synchronism, fault-detection, and fault-tolerance, while effectuating loose coupling of the processes.

If the processes achieve comparable states following completion of the first and second sequences, the first process takes up execution of a third sequence, while the second process takes up execution of a fourth sequence. In the event that one of the processes does not complete its respective instruction sequence, or if the process states do not otherwise favorably compare, an error is signaled.

Alternatively, or in addition, the method calls for rolling back the processes to performing error diagnostics and/or rolling back the processes to their most recent respective states of agreement, e.g., the states prior to execution of the first and second sequences, and retrying execution of those sequences.

Further aspects of the invention provide methods as described above in which the instruction sequences are selected for execution based on the process states. As above, the state information can include registers, interrupts, memory, flags and/or events in each of the respective processes, though one aspect of the invention calls for the selection to be made based on the tasks queued for completion in each process (e.g., as a result of interrupts received during execution of the prior sequences). Selections made for the respective processes, e.g., the identity of the aforementioned third and fourth sequences, can be compared between the processes. As above, the comparison can be conducted over a predetermined interval and, in the event of non-agreement, an error can be signaled and/or the processes rolled back to a prior state.

Still further aspects of the invention provide control apparatus operating in accord with the above-described methods. These can comprise devices at any level of the process control hierarchy, from process control field device (e.g., sensors) to block controllers to process controllers to plant and enterprise servers.

Still other aspects of the invention provide general purpose digital data processing apparatus and methods paralleling the control apparatus and methods described above.

These and other aspects of the invention are evident in the drawings and in the description that follows.

Systems according to the invention have a number of advantages over the prior art. Among these are that they permit control devices to provide fault-detection and achieve fault tolerance without special-purpose hardware. Instead, only a minimal facility to exchange state information between the modules is necessary. The invention, as evident below, can be used with either hardware-based error

detection or software comparison or arbitration. The invention can also be embodied in dual-partnered systems, as well as those that are triply or quadruply redundant.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a pair of process control field devices according to the invention;

Figure 2 depicts a method of operation of a synchronizing-scheduler in the devices of Figure 1;

Figures 3A and 3B depict a method of operation of a synchronizer in the devices of Figure 1; and

Figure 4 depicts a process control system with field devices, control processors and plant or enterprise servers according to the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a pair of process control devices 10, 12 according to the invention. The illustrated devices are field devices, though the invention may applied to other control devices as well, such as, by non-limiting example, block controllers, process controllers, plant and enterprise servers, and environmental and industrial controls and controllers.

Illustrated devices 10, 12 include processing sections 18, 20 and sensing apparatus -- in this case, by way of non-limiting example, sensors 14, 16 for measuring flow along pipe 17. The sections 18, 20 are constructed and operated in accord with the teachings herein to provide fault detection and fault tolerance. They can also be configured to provide signal conditioning and other functions commonly known in the art. To these ends, illustrated sections 18, 20 can comprise general or special purpose digital data processing apparatus, including central processing units 22, 24 and various storage devices, such as code stores 26, 28 and event stores 30, 32. These storage devices may be embodied in any conventional storage units, such as, by way of non-limiting example, EEPROMS or DRAMS, and they may be implemented in any conventional manner, e.g., arrays, linked lists, b-trees, and so forth. In addition to, or in lieu of such digital data processing apparatus, illustrated sections 18, 20 may comprise analog electronics or other functionality configured for carrying out the tasks described herein.

Control devices 10, 12 are coupled to one another for the exchange of state information of the type described below. In the illustrated embodiment, such exchange takes place over communication pathway 38, which may be connected to the serial, parallel, or other input/output ports of the respective processing sections. Pathway 38 can be implemented via cable, bus, LAN, WAN, Internet, or other direct, indirect or networked medium. Moreover, it may be implemented via conventional cabling or via infrared, microwave or transmission-based medium.

In addition to being coupled to one another, sections 18, 20 can be coupled to other elements of a process control system, such as, block controllers, process controllers, and plant and enterprise servers. In the illustrated embodiment, such coupling is provided via network 40 (e.g., an Ethernet network). As above, this may be implemented via cable, bus, LAN, WAN, Internet, or other direct,

indirect or networked medium, using conventional cabling, infrared, microwave or transmission-based medium. Those skilled in the art will, of course, appreciate that the information exchange described below as taking place over line 38 can, instead or in addition, take place over network 40.

Illustrated stores 30, 32 contain information reflecting a state of each respective process control device 10, 12. In the illustrated embodiment, these comprise event flags generated by interrupt handlers (not shown) executing on the respective central processing units 22, 24. The flags represent external interrupts, such as those received from sensors 14, 16, from other process control devices coupled to network 40, and from other peripheral attached directly to devices 10, 12. The flags also represent internal interrupts, such as those generated by watch-dog and other timers (not shown) internal to the processing sections 18, 20.

The state of control devices 10, 12 may be reflected by other information, as well. This can include, for example, pending interrupts, task and event queues, and so forth. As discussed below, state information used in the illustrated embodiment also includes the identity portions of code sequences, i.e., subsequences or slices, elected for execution by each device 10, 12, which information may be maintained, for example, in registers or other memory areas in the devices.

Code stores 26, 28 contain computer instructions ("code") for execution by processing sections 18, 20, e.g., in response to events listed in event stores 30, 32. The code, which may of the type generally known in the art of process control or which may be otherwise suitable therefore (whether or not currently known), is preferably parceled into units or "slices" 42 - 52. These slices may be, for example, indivisible or "atomic" sequences of code. They may also be divisible sequences sized for efficient execution in view of the synchronization methodologies described herein. As discussed below, a preferred slice represents an increment of work that is small enough to be completed in a time interval consistent with the delay that can be accepted if the thread executing it is preempted.

Illustrated central processing units 22, 24 are of the conventional type known in the art capable of executing instruction sequences within processes, threads and other such instruction execution contexts (collectively, "processes"). In addition to slices 42 - 52 contained in code stores 26, 28, the

CPUs 22, 24 execute code for purposes of identifying events and scheduling slices (i.e., synchronizing-schedulers 54, 60), synchronizing event identification and slice selection (i.e., synchronizers 56 62), and dispatching slices for execution (i.e., dispatchers 58, 64).

In the illustrated embodiment, code executed by the CPUs 22, 24 and, particularly, code executed from stores 26, 28, is broken up into execution slices 42 - 52. An execution slice is an increment of work that is small enough to be completed in a time interval consistent with the delay that can be accepted if the process or thread executing it is preempted. In addition, each slice starts at the logical beginning of an operation, and ends when the operation is completed. At slice completion, the execution is between contexts.

While this approach can be applied to general purpose software, it is particularly suited to applications that have an appropriately "sliceable" structure. Process control software has this characteristic. Exploiting it accomplishes two things. The first is to reduce the "operating system" to a simple synchronizing-scheduler. The second is that the synchronizing-scheduler 54, 60 simplifies attaining fault tolerance. It is this simplification, among other things, which makes it possible to support fault tolerant operation with the available hardware resources.

Each synchronizing-scheduler 54, 60 recognizes and synchronizes all pending events. As discussed above, events typically result from interrupts, e.g., clock interrupts, communication interrupts, etc., that have occured during execution of a prior slice. Some events, indeed, are a natural result of slice execution itself, e.g., events reflecting the availability of data computed by the prior slice. After each synchronizing-scheduler 54, 60 synchronizer its pending events with the other synchronizing-scheduler (or, other synchronizing-schedulers), it selects a next slice for execution, and synchronizes *that* selection. In some embodiments, the identities of the slices selected for execution are not synchronized; rather, merely, the fact that slices have been selected for execution.

Operation of the illustrated synchronizing-schedulers 54, 60 is shown in Figure 2 and described in the pseudo-code fragment that follows:

while true (step 70)

```
if rollback_cnt > max_rollbacks break; (step 72)
if more unsynchronized events then (step 74)
        select highest priority pending event (step 76)
        invoke synchronizer to confirm that partner selected same event (step 78)
        if synchronization was successful (step 80)
                rollback_cnt == 0 (step 79)
        else
                increment rollback_cnt (step 81)
        endif
        continue
endif
select code slice for execution to process event (step 82)
invoke synchronizer to confirm that partner selected same slice (step 84)
if synchronization not successful (step 86)
        increment rollback_cnt (step 81)
        continue
endif
rollback_cnt == 0 (step 87)
dispatch to execute selected slice (step 88)
end while
signal error (step 91)
perform error handling (step 92)
```

The type and degree of error handling in step 92 depends on the nature of the application. In some embodiments, no error handling at all is performed after the fault is signalled. In other embodiments, the processes are rolled back to state of last agreement. In still other embodiments, a fault diagnostic procedure is performed to isolate the source of error.

As noted above, devices 10, 12 on both sides of the redundant pair have synchronization, or sync, lines 38 running between them that are used to keep their operation synchronized. This enforces loose coupling between the devices 10, 12 and, more particularly, between their respective processing sections 18, 20. One of the devices 10 is configured to be the master, and the other 12, the shadow. The synchronizer operation for the master 10 is shown in Figure 3A and described in the pseudo-code that follows:

```
write event/slice code on sync out lines (step 102)
set timeout register
initialize sync results to false
```

```
do
        decrement timeout register
        read input from partner (step 104)
while shadow agrees = false and timeout != 0 (step 106)

if timeout register != 0 (step 108)
        write "null" code on sync out lines (step 110)
        do
                decrement timeout register
                read input from partner (step 112)
        while shadow agrees = true and timeout != 0 (step 114)
endif
if timeout register != 0 (step 116)
        sync results = true (success) (step 118)
endif
return sync results
```

The shadow 12 executes a similar synchronization routine, which is described below and shown in Figure 3B.

```
set timeout register initialize sync result
to false
do
        decrement timeout register
        read master synch state
while shadow state != master state and timeout != 0
operation code and timeout register != 0
if timeout register != 0
        write "shadow agrees" on synch line
        do
                decrement timeout register
                read input from partner
        while master state != "null" and timeout != 0
endif
if timeout register != 0
        synch results = success
endif
return synch results
```

As noted above, the invention can be applied to process control apparatus other than field devices 10, 12, as well. A process control system in which the invention is employed in a plurality of pairs of process control devices 120, block controllers 122 and plant servers 124 is illustrated in

Figure 4. Although the processing performed by these various apparatus differs in accord with their levels in the system, their processing sections perform scheduling and synchronization in the manner describeds above in order to provide fault-detection and tolerance.

In one embodiment, the invention is employed in coprocessors that serve as communication controllers for a network of field modules. Such coprocessor can be contained, for example, within one or more of the block controllers 122 of the type shown in Figure 4. Software executing on the CPUs of those coprocessors is both multithreaded and fault tolerant. It constitutes a complete stand-alone executable that sits directly on top of the hardware, preferably, without the need for a separate operating system.

The aformentioned embodiment is multithreaded with priority scheduling. It does this without a general purpose context-preserving task switch mechanism. Instead the work is broken up into execution slices as described above. The basis of the slices chosen for the coprocessors is to associate a slice with the work required to send or receive a particular message over network 40. There are two reasons for this. First, the hardware events associated with the network activity are good synchronization markers. Second, when one considers fault tolerant error recovery scenarios, the last completed network message is a good point to restart from.

Within the aforementioned coprocessors, each channel is an independent stream of communications transactions associated with a set of data structures in memory that they share between them. A channel is also an execution thread. Each channel has a unique priority. On the "host" side of the shared memory interface, each channel is associated with a vrtx task of corresponding priority.

The communications channels are identical except for priority and assigned buffer sizes. The number of channels is controlled by the initialization of master control structures contained in the shared memory. They share a single common set of routines that execute three basic types of slices:

1. routines that convert a list of transaction requests into the corresponding request messages;

2. routines that send the messages and wait for the replies;

3. routines that convert the messages into the required lists of responses

The execution state of each channel is recorded in a channel data structure within the shared memory. Once started, execution of the slice continues until it completes. The stored state information identifies the next slice to be executed, if any, for each channel.

Fault tolerance error scenarios include exception cases where a slice in progress cannot be completed. In these cases, it is simply abandoned. Because the channel data that controls scheduling is not updated until the slice completes, the slice will be restarted from its beginning when/if the execution thread is resumed.

At a logical and priority level above the communications channels, the master control structures provide what can be thought of as a master control channel. Associated actions at this level always complete. They are independent slices. They range from bus switching to mechanisms for fault tolerance exception handling. Each synchronizing-scheduler within the coprocessors checks the master control structure, then the channel information in priority order, to identify the highest priority slice available for execution. Interrupts are serviced during slice execution, but any resulting thread switch occurs after the current slice is completed.

After a synchronizing-scheduler selects the next slice, it invokes its respective synchronizer. During married fault tolerant operation, the synchronizers in the master and shadow use hardware handshake lines and logic as described to ensure that the both sides choose the same next slice. If this process fails, the synchronizers proceed as described above.

In this coprocessor embodiment, the sync lines transmit data with the following values and meanings:

0. NULL (no sync state being declared)
1. Entering sched
2. Host interrupt
3. Transmit sync
4. Receive event
5. Time event
6. Interrupt to Host
7. Error case

Continuing discussion of the aforementioned embodiment, the synchronizer must be able to unambiguously establish success or failure and it must always return the same value on both sides. To this end, the hardware handshake in the coprocessor uses three parallel lines from master to shadow and 1 from shadow to master. The logic above has a timing window. If the shadow sets the "shadows agress" true just as the master times out, the master will return failure and the shadow will return success. The master mus to continue to assert the synch code for a brief intercval after it sees the shadow agrees signal. The shadow then checks that the masters synch code is set to null slightly after the agree line goes true. Those skilled in the art will, of course, appreciate that operation of any given synchronizer will depend strongly on the number of different states that require recognition and the hardware resources available to support it.

Described herein are methods and apparatus meeting the desired objects. Those skilled in the art will appreciate that the specific embodiments shown in the drawings and described above are examples of the invention and that other embodiments incorporating changes therein also fall within the scope of the invention. Thus, for example, it will be appreciated that the invention can be applied to process, industrial, environmental and other control systems and devices utilizing triple modular redundancy, quaduple modular redundancy and still higher levels of redundancy, as well as to those utilizing paired units (or double modular redundancy) as shown in the examples herein. By way of further example, it will be appreciated that the partnered processes can exchange and compare state information different than that described here. By way of still further example, it will

be appreciated that the invention can be applied to digital data processing apparatus and systems, as well.

In view of the foregoing what we claim is: